WHAT DATA MODELS CAN'T DO

David C. Hay

I USED TO THINK . . .

Your author discovered data models in the early 1980's, without recognizing that they were anything other than a convenient way of describing a data base structure. Then, in the mid-1980's, he encountered Oracle's approach and discovered them to be very powerful in their ability to describe the structure and important aspects of a business. They proved to be much more expressive and useful as an analysis tool than data flow diagrams or anything else that had been available.

He got in trouble with developers, though, by thinking thatt analysis was finished when the data models were done. The developers, it turns out, still had to do a second analysis to find out what the ***business rules*** were. *When* are occurrences of this created? What conditions had to be met? How do you determine which account to charge? And so forth.

The data model clearly isn't enough. What it lacks is a formal way of describing the business rules that control the business (and therefore the model).

The information industry is beginning to formalize this shortcoming. The IBM User Group, GUIDE has established a project to articulate the requirements and possible approaches to specifying business rules. Ron Ross has written a comprehensive book on the subject.[1] Barbara von Halle has written a series of three articles for *Database Programming and Design*.[2] Oracle UK is doing research to determine how formal documentation of business rules might be incorporated into future CASE tools.

This paper will describe some of the shortcomings of data modeling in describing business rules, and present some approaches to making up for these shortcomings.

PROBLEMS WITH DATA MODELS

A data model for the most part only describes the *structure* of information. It says very little about how that information can or must be used. Some rules are included: any "must be" relationship is a rule. Definitions are rules. The fact that an occurrence of a sub-type must be an occurrence of a super-type is a rule. That's about it.

---

1 Ronald G. Ross, *The Business Rule Book*, (Boston:Database Research Group, Inc., 1994).

2 Barbara von Halle, *Database Programming and Design*, "Back to the Business Rule Basics", October, 1994, "Living by the Rules", November, 1994, "Lessons to Learn from Tee-ball", December, 1994.

There are at least four categories of things that cannot adequately be described in data models:

- ✓ Implied assumptions
- ✓ When Optional relationships are activated
- ✓ How to keep multiple paths between entities consistent
- ✓ Derivations

These are described further, below.

Implied assumptions

Some business rules are so obvious, that we just assume they have been asserted, when in fact they have not. Figure 1. for example, shows a simple case of recursion.

The model says that each ORGANIZATION may be *composed of* one or more ORGANIZATIONS, and that each ORGANIZATION may be *part of* one and only one ORGANIZATION. What it does not say, however, is that there is any restriction on the ability to specify that an ORGANIZATION may be *part of* itself. That is, you could, under the terms of this model, say that Denver South High School is part of Denver South High School. More subtly, you could say that Denver South High School is *part of* the Denver School District, which in turn is *part of* Denver South High School.

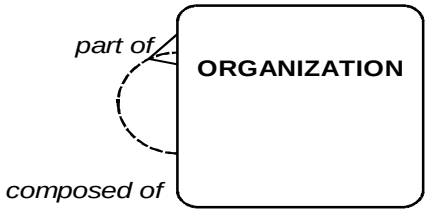

Figure 1: Recursion Assumption[3]

The model does not prevent such loops.

Related to this is the example shown in Figure 2.

This model expands on the previous one, adding the notion that an ORGANIZATION is an example of an ORGANIZATION TYPE. That is, "the shipping department" is an example of a "department", while "Coca Cola" is an example of a "corporation". Just as an ORGANIZATION may be *composed of* one or more ORGANIZATIONS, as stated above, so too an ORGANIZATION TYPE may be *composed of* one or more ORGANIZATION TYPES.

What assumed but not stated, however, is that the two hierarchies are consistent with each other. We assume, for example, that if "the shipping department" is *an embodiment of* the ORGANIZATION TYPE

[3] The models in this paper are from: David C. Hay, *Data Model Patterns: Conventions of Thought*, (New York: Dorset House Publishers, 1996)

"department", and that a "department" is always *part of* a CORPORATION and always *composed of* one or more "sections", then "shipping department" will be *part of* "Coca Cola" (a "corporation") and not the other way around. That is, the assumed rule is that the structure of ORGANIZATION TYPE is consistent with the structure of ORGANIZATION.

But that rule cannot be stated in our model.

Optional relationships don't tell you when

Another shortcoming of data models when it comes to describing business rules is the inadequacy of optional relationships. These say that an occurrence of one entity *may* be related to one or more occurrences of another, but they do not say when and under what circumstances

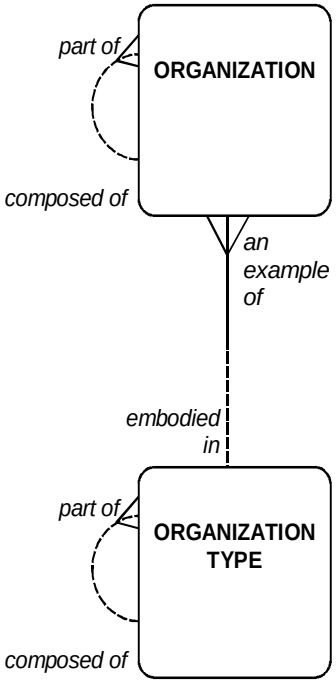

Figure 2: Related Recursions

For example, Figure 3, a physical ASSET may be *accounted for in* a financial ASSET ACCOUNT. The question is, when? While it is typical for the accounting to lag a while

after the acquisition of the ASSET, there is probably a policy which says that (for example) within one calendar month of its acquisition, each ASSET **must be** *accounted for in* one and only one ASSET ACCOUNT. This cannot be shown here.

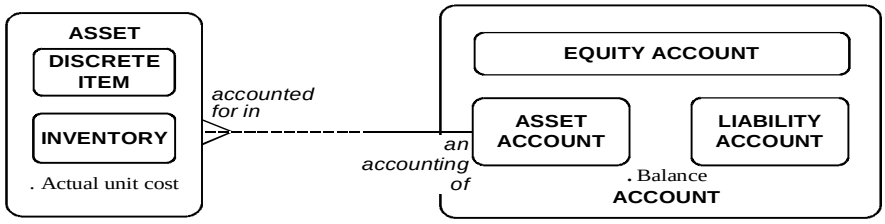

Figure 3: Asset Usage

The same problem, but with a different twist, is shown in Figure 4. Here, normally, a WORK ORDER is *carried out via* one or more ACTIVITIES. Similarly, most of the time, an ACTIVITY is *authorized by* a WORK ORDER. The problem is that under certain circumstances, an ACTIVITY may be carried out that is not *authorized by* a WORK ORDER. Under what circumstances is that permitted? The model doesn't say.

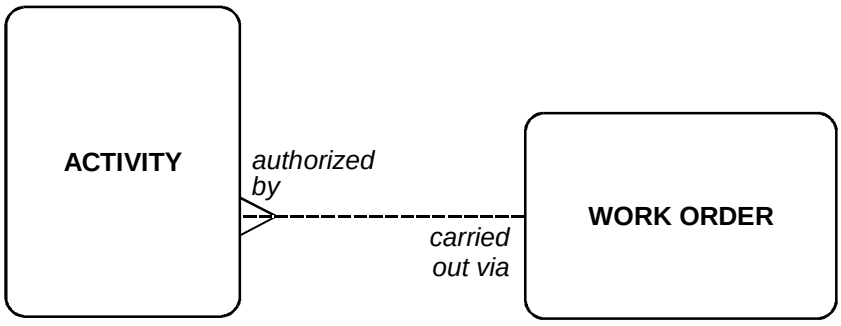

Figure 4: Activities and Work Orders

Multiple paths require care

Another area where data models cannot express all that must be said is probably the most treacherous. Any time it is possible to navigate from one entity to another by means of two paths, it is possible to specify combinations of occurrences that are nonsense. In each case, a business rule is required to specify what combinations are valid.

For example, in Figure 5. a TEST REQUIREMENT *from* a particular SAMPLE METHOD is *for* a particular TEST TYPE. Meanwhile, each TEST must be *conducted on* a SAMPLE, which, in turn, must be *drawn according to* a particular SAMPLE METHOD.

There is nothing in the model, however, to assure that, given a SAMPLE METHOD used, the TESTS conducted are *examples of* the TEST TYPES that are *the object of* TEST REQUIREMENTS *from* the same SAMPLE METHOD. The SAMPLE METHOD may require one set of TEST TYPES, but the actual TESTS conducted may be completely different.

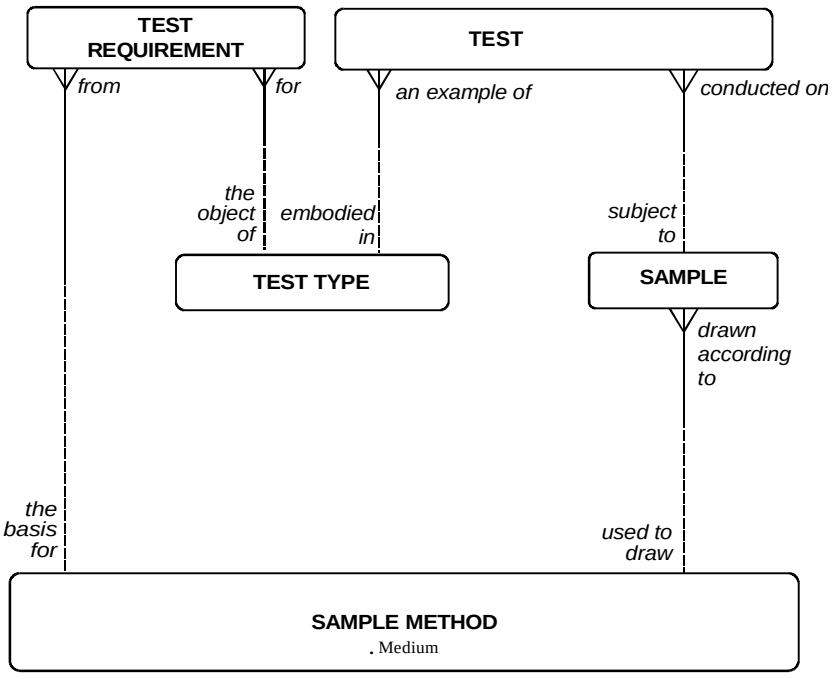

Figure 5: Test Requirements and Tests

Note that this raises an important point: It may be appropriate to bounce any data entered that violated this condition. If in the world, however, it sometimes happens that the wrong tests are given, this model is exactly correct. There may be a business policy not to, in which case, a system based on this model should alert the community

that the wrong things are happening, but it should not necessarily prevent these wrong things from being recorded.

Note the distinction between enforcing business rules by constraining what is allowed to be entered, and enforcing them in the world by notifying people when they are being violated in the world.

In some cases inconsistencies are allowed. For example, in Figure 6. It is possible for a PERSON to be in an EMPLOYMENT with one ORGANIZATION, while h' POSITION ASSIGNMENT may be *to* a POSITION *defined by* another ORGANIZATION. This is reasonable in the case where someone is hired in one department, but temporarily assigned to another.

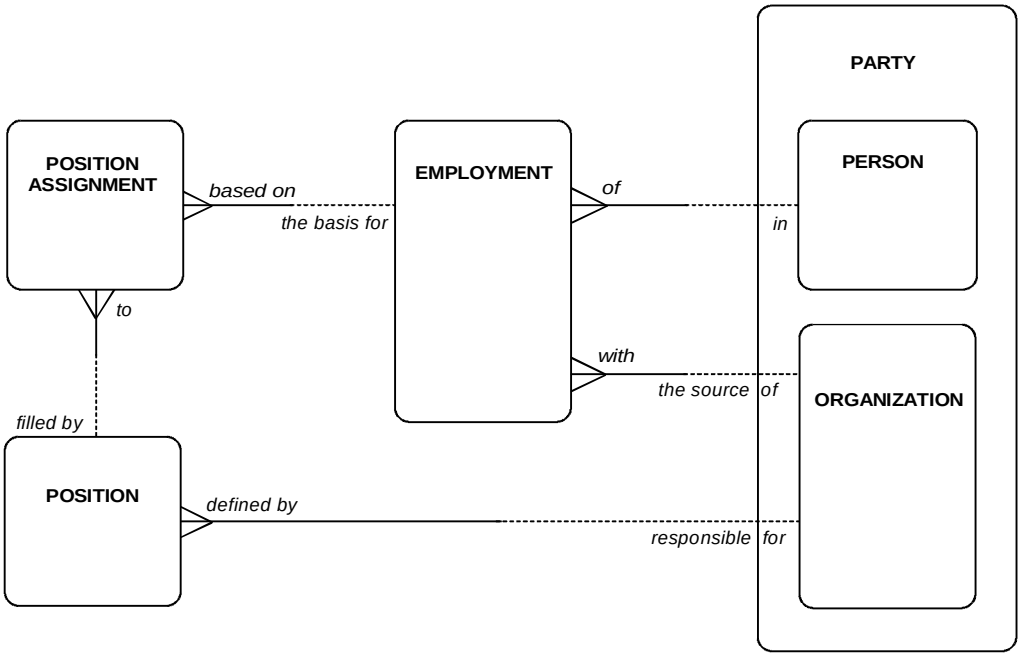

Figure 6: Employment

In some cases, constraints can be represented in a data model. Figure 7 is a variation on Figure 5. In this case each test must be the carrying out of a test requirement. This at least makes it impossible to specify a test that is not required by somebody. Unfortunately this still does not require the sample method which is the basis for the test requirement to be the same as the sample method which was used to draw the sample which was subject to the test.

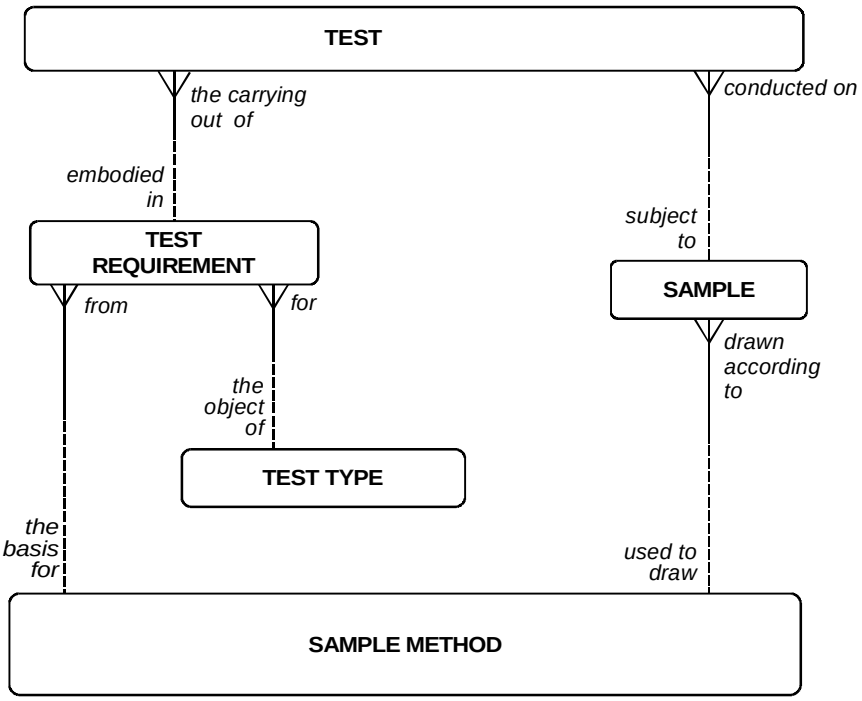

Figure 7: Test Requirements, again

Fundamentally, the data model cannot describe constraints among specific occurrences of entities. It can only describe constraints among the entity types.

No provision for derivations

A final shortcoming of data models, at least as they are usually drawn, is lack of attention to derived data.

Relational theory does not provide for computed columns, and data modeling has always tended to reflect this. In fact, however, using derived attributes is a very powerful way to present the logic behind complex database-wide calculations.

For example, Figure 8 shows the model for TIME SHEET ENTRY, the vehicle for recording the amount of time spent by a PERSON on either an ACTIVITY or a WORK ORDER. The CASE*Method as formally defined (and the CASE tools which support it) does not provide a formal way of documenting how labor costs are derived from the information in the model.

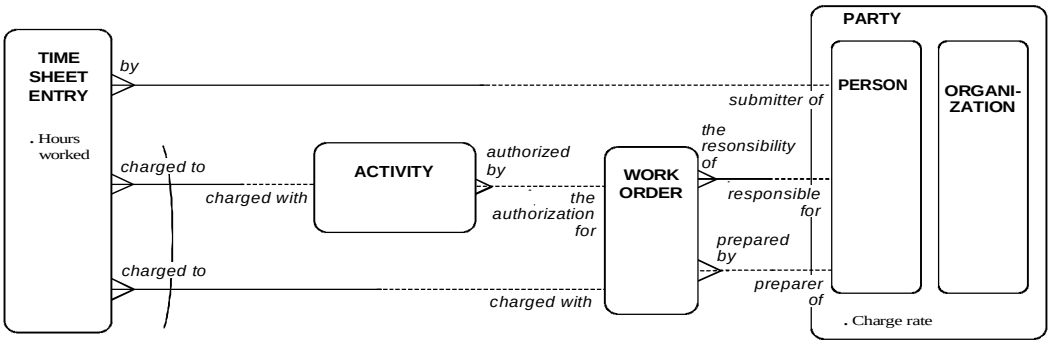

Figure 8: Labor Costs

BUSINESS RULES

What then, is a business rule? According to the *Business Rule GUIDE Project*, it is a "definitional, structural, or conditional constraint upon the business."

Barbara von Halle defines four kinds of business rules:[4]

- ✓ ***Definition*** — A business noun's meaning or significance. These are usually documented behind a data model.
- ✓ ***Fact (type)*** — An assertion that a particular noun has a property or plays a role, or that one or more objects (nouns) participate together in a relationship. These relationships are what a data model is mostly about.
- ✓ ***Constraint (type)*** — A restriction of or the validation rule about a fact type's population. These are statements about the business in the form of "must be" or "must not be". *Static constraints* describe what may be so in any state of the database. *Dynamic constraints* restrict the possible transitions between database states. These should be documented along with the data model and often are not. A way of describing these on the data model drawing itself would be welcome.
- ✓ ***Derivation specification (type)*** — a mechanism for deriving new fact types from existing fact types.

("Type" may be used since we are talking here about the *definition* of a fact, constraint, etc., not each occurrence of it.)

DEFINITIONS

What do we mean by PARTY? CONTRACT? EXPENSE ALLOCATION? The definition of an entity will place constraints around what may or may not be a legal occurrence of that entity. Typically, definitions are delivered along with a data model, to define the entities which appear in that model.

---

[4] Barbara von Halle "Living by the Rules", *Database Programming and Design*, October, 1994, with contributions from G. M. Nijlssen and T. A. Halpin, *Conceptual Schema and Relational Database Design*, (Sydney:Prentice Hall, 1989)..

FACTS

An entity model describes facts about a business. Specifically, a data model can represent the following kinds of facts:

- ✓ Each occurrence of an entity may or must have "the property of"... That is, it may be described by an attribute.
- ✓ Each occurrence of an entity may or must be "a kind of"... That is, it is a sub-type of — encompassed in the meaning of — another entity.
- ✓ Each occurrence of an entity may or must "play the role of". That is, it is related in some way to another term. Some relationships are particularly common, such as:
    - Each occurrence of an entity may or must be "composed of" another enitity.
    - Each occurrence of an entity may or must be "the embodiment of" another, type, entity.

There are, of course, many others.

CONSTRAINTS

Constraints, in general, are not handled well by data models. While data models can describe what "may be" true, they can only describe a few kinds of things that "must be" or "must not be" true. This is true of all flavors of data modeling except NIAM, which does include an extensive facility for describing constraints.[5]

As described above, Ron Ross has developed a comprehensive approach to documenting the constraints that would apply to a data model — graphically, on the data model itself.

He sees two basic kinds of rule elements:

- ✓ One describes an ***integrity constraint.*** This is something that *must be true* about an entity relationship or attribute, by definition.
- ✓ The other describes a ***condition.*** This may be true or false. Depending on the condition other constraints may apply.

Each rule describes the effect of a ***constraining object*** upon a ***constrained object.*** On the left side of Figure 9 is an *integrity constraint*, represented by the arrowhead with the "X" in it. (This is a modification to Mr. Ross's notation, to adapt it to the CASE*Method. In his original notation, attributes are shown in circles, outside the entity box.) An integrity constraint must be true. The constraint shown says that any occurrence of CUSTOMER must have a value for the attribute "address". CUSTOMER is the constrained object, and "address" is the constraining object. The "X" in the symbol is one of 42 ***rule types*** that Mr. Ross describes. It indicates the rule type "mandatory". That is, it asserts that the constrained object "must have" an occurrence of the constraining object.

On the right side of Figure 9 is a *condition.* This says that *if* an occurrence of CUSTOMER has a value for the attribute "address", then carry out the next part of the rule. Note that rule elements can indeed be strung together.

---

[5] The only good book in English describing NIAM is Messrs Nijssen and Halpin, referenced above.

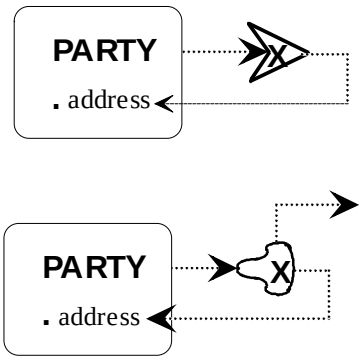

Figure 9: A Business Rule

Each of the 42 rule types may be used as an integrity constraint or a condition to describe a particular situation. Mr. Ross contends that this list of rule types is atomic and fundamental. While extensive, Mr. Ross does not claim that the list is exhaustive. There are undoubtedly more to be discovered. The list is divided into nine categories:

- ✓ ***Instance verifiers***, requiring an occurrence of an entity to be present at the creation of, during the life of, etc. an occurrence of another entity.
- ✓ ***Type verifiers***, requiring occurrences of entities to be mutually exclusive, mutually dependent, etc.
- ✓ ***Sequence verifiers,*** requiring occurrences to be created in a particular order.
- ✓ ***Position selectors***, requiring reference to the lowest value of an attribute, or the highest value, or the earliest, etc.
- ✓ ***Functional evaluation,*** requiring occurrences of an entity to be unique, ascending, fluctuating, etc.
- ✓ ***Comparative evaluators*** allow for comparisons (less than, equal, etc.) between the attributes of two occurrences of the same or different entities.
- ✓ ***Calculators***, are used in a rule to derive values used by the rule.
- ✓ ***Update controllers*** determine whether a value must be forever fixed, updatable, etc.
- ✓ ***Timing controllers*** control the timing of events.

These rule types can be applied, one at a time or in groups, to the data models described above, thus adding the rules necessary to complete them:

Implied assumptions

The problem of ORGANIZATIONS being *composed of* themselves is handled by the inverse of a "mandatory" rule. The comparative evaluator condition "EQ"(ual) establishes that under certain circumstances the "names" of two occurrences of ORGANIZATION will be equal. (The circle with the slash negates the "EQ" rule, making it "must not be".) Specifically, the mandatory integrity constraint, "X" requires this to be true. (Or rather, not true, given the negation qualifier.)

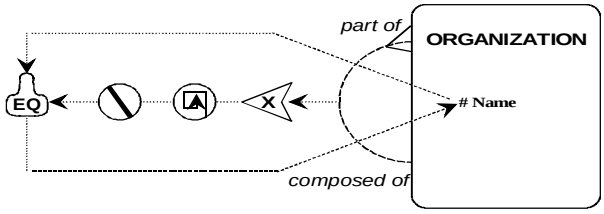

Figure 10: Recursion Assumption

The multi-level hierarchy is handled by a "restrictive" constraint. (See Figure 11.) This means that an occurrence of the relationship being constrained may only exist if specified conditions are met. Specifically, the numbered circles require that the other relationships be traversed in sequence to arrive at the same entity occurrences as those in the constrained relationship. In this case, the constraint says that the *composed of* relationship is constrained by three navigation steps. If you want to say that one occurrence of ORGANIZATION is *part of* another, first look at which ORGANIZATION TYPE this occurrence is *an example of.* Second, look to see which ORGANIZATION TYPE the ORGANIZATION TYPE is *part of.* Third, look to see theoccurrences of ORGANIZATION that this second ORGANIZATION TYPE is *embodied in.* The occurrence of ORGANIZATION on the *part of* side of the new relationship has to be one of those.

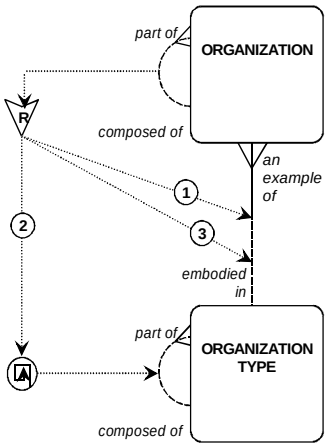

Figure 11: Related Recursions

Optional relationships

The ASSET / ASSET ACCOUNT example of optional relationships is addressed in Figure 12. This rule makes use of the *condition* symbol (the one that looks like a bicycle seat). The particular rule type is "TI" which means "elapsed time". Its qualifiers are "L" which means set the time as a lower limit, and "1 wk", which is the amount of time involved. Specifically, this is a test to see if at least one week has passed since the creation of an occurrence of ASSET. Since it is a condition, if it tests for true, another rule is applied — in this case the "X" for "mandatory". If at least one week is passed, an occurrence of *accounted for* must be created. That is, each ASSET must be *accounted for* by one and only one ASSET ACCOUNT within one week of the creation of the ASSET.

Figure 13 shows our other example of an optional relationship. In this case, it was first necessary to expand the model to clarify the terms required to establish the constraint. It turns out (in this imaginary world) that if an ACTIVITY is not *authorized by* a WORK ORDER, it must be *authorized by* a PERSON. That person may be the *holder of* a particular POSITION. We want to constrain the "*authorized by* one PERSON" relationship, so that it can only be established if the PERSON is *holder of* a POSITION whose "name" is "supervisor".

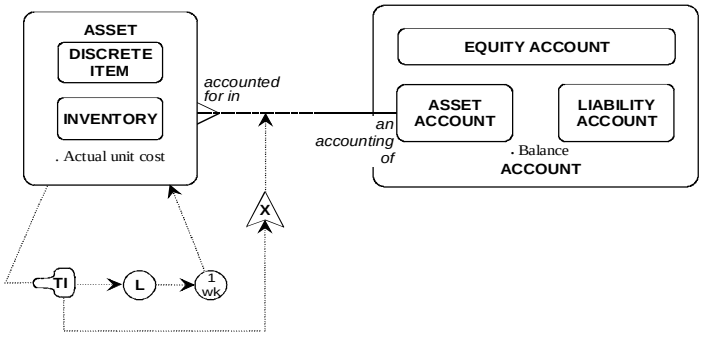

Figure 12: Asset Usage

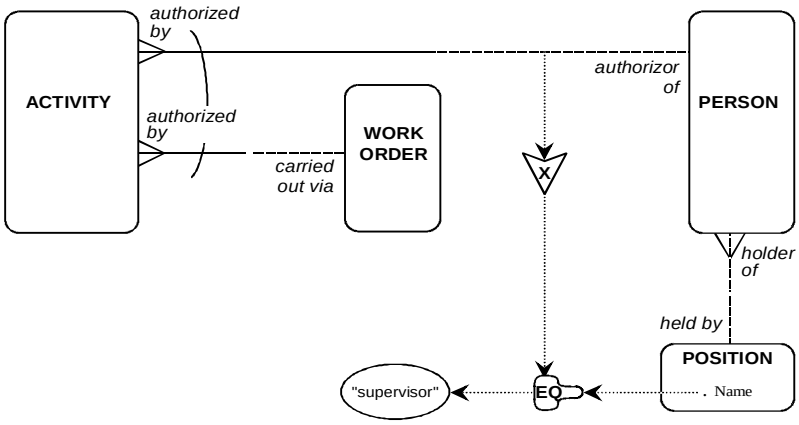

Figure 13: Activities and Work Orders

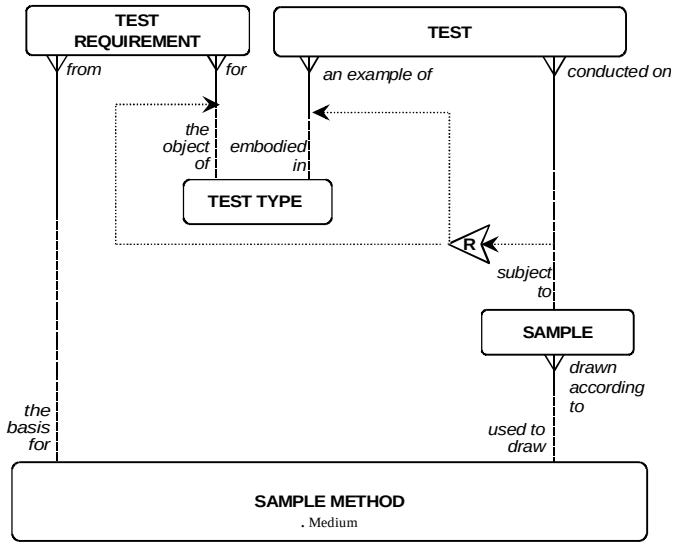

Figure 14: Test Requirements and Tests

To do this, we say that an occurrence of the relationship can only be established if the "equal to" condition pointed to by the "X" is true. That is, the condition asking whether the "name" attribute of an occurrence of POSITION is "EQ"(ual to) the value "supervisor" must be "true".

That is, an ACTIVITY may be *authorized by* a PERSON only if that PERSON is *holder of* the POSITION of "supervisor".

Multiple paths require care

The "restricted" rule type is specifically intended to deal with multiple relationships between the same entities. In our first example (Figure 14), the establishment of a TEST *conducted on* a SAMPLE is restricted to those SAMPLES whose SAMPLE METHOD is *the basis for* a TEST REQUIREMENT *for* a TEST TYPE that is *embodied in* the TEST.

The "R" type integrity constraint constrains the relationship showing the SAMPLE that a TEST is *conducted on* with the relationship specifying the TEST TYPE that TEST is *an example of, and* the relationship specifying that a TEST REQUIREMENT *for* that TEST TYPE is *from* the SAMPLE METHOD *used to draw* the SAMPLE.

As stated above, in some cases inconsistencies are allowed. In Figure 6 there is no need to add any constraints, since it is permitted for the *defined by* relationship to point to a different ORGANIZATION than that which the EMPLOYMENT is *with.*

Adding constraints in the data model itself does not significantly affect the constraints described by this notation. In Figure 16 (a variation on Figure 14, where TEST REQIREMENT has been inserted between TEST and TEST TYPE), the "restricted" rule type can be applied to the one relationship showing that a TEST is *the carrying out of* a TEST TYPE. As in Figure 14, this says that a TEST may be *carried out on* a SAMPLE, only if it is *the carrying out of* a TEST REQUIREMENT which is *from* the same SAMPLE METHOD that the SAMPLE is *from.*

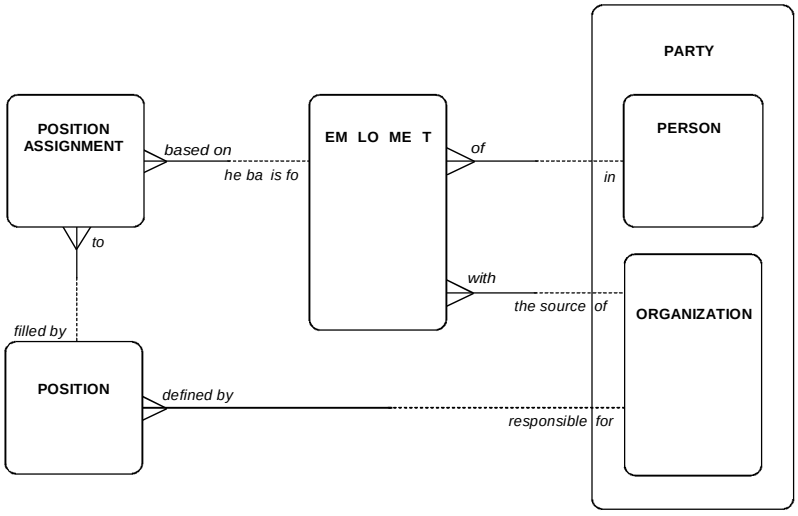

Figure 15: Employment

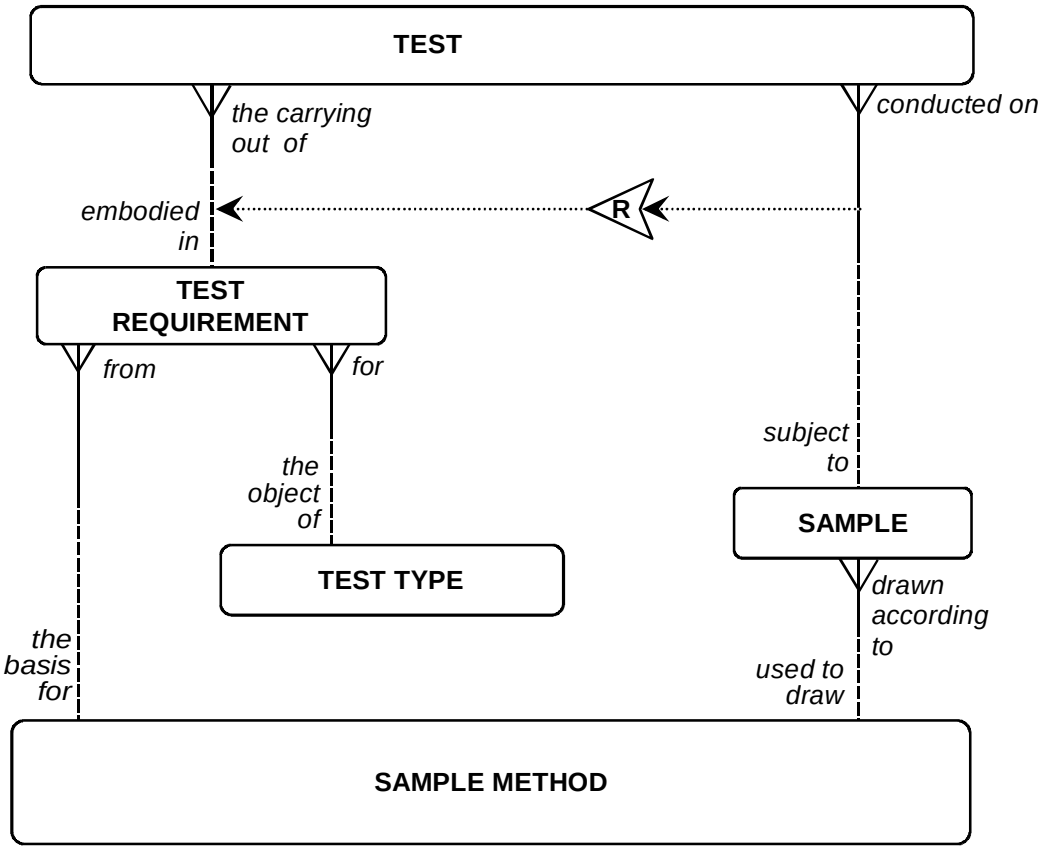

Figure 16: Test Requirements, again

No provision for derivations

Mr. Ross' notation provides for calculated field values. There are three constraint types ("SUM", "SUB", and "MULT") in his "Calculators" rule category. It is not necessary to use these, however, to show derivations on a data model. Figure 17 shows how derived attributes can be represented on the model itself. The calculations are not shown on the drawing, (as they are when Mr. Ross' notation is used), and unfortunately the CASE*Designer allows only derived attributes to be specified as text.

In our example, since "charge rate" is an attribute of the PERSON who is the *submitter of* the TIME SHEET ENTRY, it is possible to define the "cost" of the time worked as "hours worked" (in TIME SHEET ENTRY), times "charge rate" (in PERSON, inherited from PARTY).

Once the "cost" has been calculated for one TIME SHEET ENTRY, it is a simple matter to sum that attribute across all the TIME SHEET ENTRIES *charged to* an ACTIVITY, in order to determine the "labor cost" of the activity. The sum of the "labor costs" of all the ACTIVITIES *authorized by* a WORK ORDER, plus the sum of the "costs" of all the TIME SHEET ENTRIES directly *charged to* the WORK ORDER yields the "total labor cost" of the WORK ORDER.

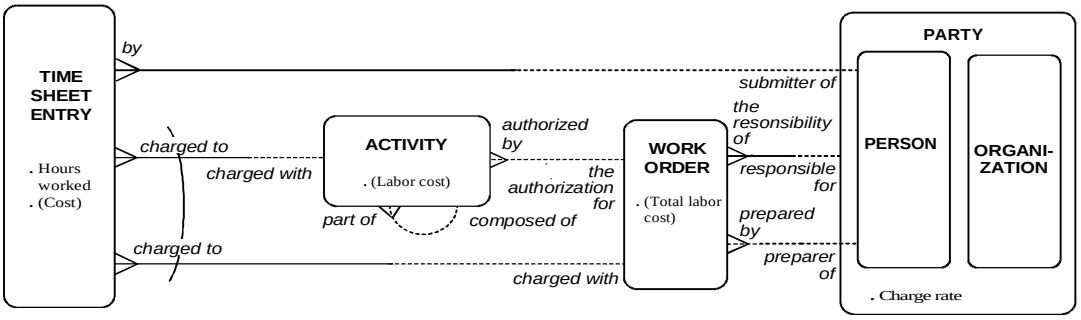

Figure 17: Labor Cost

About the Author . . .

A twenty-five year veteran of the Information Industry, Dave Hay has been producing data models to support strategic information planning and requirements planning for over eight years. He has worked in a variety of industries, including, among others, power generation, clinical pharmaceutical research, oil refining, forestry, and broadcast. He is President of Essential Strategies, Inc., a consulting firm dedicated to helping clients define corporate information architecture, identify requirements, and plan strategies for the implementation of new systems.

He is the author of the book from Dorset House Publishers, *Data Model Patterns: Conventions of Thought.* He may be reached at 713-464-8316 or *davehay@essentialstrategies.com*. His company's web page is at *www.essentialstrategies.com.*